NASA Technical Memorandum 102866

A Trajectory Planning Scheme for Spacecraft in the Space Station Environment

Jeffrey Alan Soller, Arthur J. Grunwald, and Stephen R. Ellis
Ames Research Center, Moffett Field, California

January 1991

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

Table of Contents

Abstract

Simulated annealing is used to solve a minimum fuel trajectory problem in the space station environment. The environment is special because the space station will define a multi-vehicle environment in space. The optimization surface is a complex nonlinear function of the initial conditions of the chase and target crafts. Small permutations in the input conditions can result in abrupt changes to the optimization surface. Since no prior knowledge about the number or location of local minimum on the surface is available, the optimization must be capable of functioning on a multimodal surface. It has been reported in the literature that the simulated annealing algorithm is more effective on such surfaces than descent techniques using random starting points.

The simulated annealing optimization was found to be capable of identifying a minimum fuel, two-burn trajectory subject to four constraints which are integrated into the optimization using a barrier method. The computations required to solve the optimization are fast enough that missions could be planned on board of the space station. Potential applications for on board planning of missions are numerous. Future research topics may include optimal planning of multi-waypoint maneuvers using a knowledge base to guide the optimization, and a study aimed at developing robust annealing schedules for potential on board missions.

I. INTRODUCTION

The space station Freedom will become a flexible part of the scientific community early in the next century. The purpose of this research is to develop a program which is capable of determining minimum fuel, rendezvous trajectories in the space station environment. The initial conditions of a target craft and a chase craft are input to the program. The program uses an adaptation of the simulated annealing algorithm to find a minimum fuel trajectory. This type of optimization program may assist astronauts in planning complex orbital maneuvers on board the space station.

Once the space station is fully operational, there may be several spacecraft orbiting the earth in the neighborhood of the space station. This multi-craft environment will increase the complexity of orbital maneuvers in that neighborhood. The orbital dynamics which control motion in space, are highly counter intuitive. Therefore planning missions on demand can be difficult. Currently, major orbital maneuvers are planned at Johnson Space Center before liftoff. However as the environment becomes more complex, it is unlikely that all possible mission scenarios will be envisioned and precomputed. Therefore, it would be useful to have an on-board tool which is capable of computing safe, near optimal trajectories with respect to fuel use.

The optimization scheme presented here is a second step toward developing the previously mentioned trajectory design tool. The first step was a graphic tool developed by Grunwald and Ellis [1 and 6]. The optimization is capable of finding an optimal two burn maneuver, with respect to fuel use, subject to four constraints. Those constraints are maximum arrival time, departure angle of the initiating burn, arrival angle of the rendezvous burn, and a spatial constraint which prevents flight close to the space station. Futhermore, the optimization code runs fast enough in test cases that it realistically could be used on-board the station to plan trajectories. A long term goal is to extend this optimization tool so that it is capable of planning multiple way point manuevers.

In particular, the problem that has been addressed is to find a feasible, minimum fuel, 2 burn trajectory which will initiate rendezvous between a chase craft and a given target craft. Feasible trajectories are those that do not violate any constraints. The input includes information about the object which is to be caught (the target craft) and information about the chase craft. The crafts' initial

relative positions, magnitude of relative motion, and angle of relative motion are the necessary data. If the relative time of mission initiation and rendezvous are specified, and the required input data is given, the mission fuel use is directly computable. The optimization surface of interest is the one composed of all possible combinations of initiation and rendezvous times. The point on that surface which corresponds to a feasible trajectory using minimum fuel determines the desired trajectory.

Two possible surfaces could be constructed both of which are nonlinear in general with the potential for multiple local minima. The first surface allows the time of mission initiation to vary from zero to ninety minutes, while the rendezvous time is constrained to be greater than the initiation time and less than ninety minutes. This yields a three dimensional surface with the fuel use as the dependent variable. The second surface is obtained by fixing the mission initiation time at a relative time zero, and allowing the rendezvous time to vary between 0 and 90 minutes. This is a 2 dimensional surface with the same dependent variable. The second type of surface will be considered in detail in Sections VII and VIII. Optimization of nonlinear surfaces without a prioi knowledge of local minima can be a difficult task.

The nonlinear optimization technique which has been chosen for this project is an application of simulated annealing. Simulated annealing is a stochastic procedure which has been shown to be an effective optimization technique in cases with multiple local minima. [2] The technique has been shown to be asymptotically convergent to the global solution. [4] Since convergence is achieved only after an infinite number of transitions, it is necessary to implement an approximation algorithm which is capable of locating near optimal solutions in a reasonable amount of computation time. Many other techniques are available for optimizing nonlinear surfaces. However, most require some knowledge about the surface to be optimized. In this particular application, a priori knowledge about the number or approximate location of local minima is not feasible. Those details are determined by the initial state of the system which vary from mission to mission. It is possible that one surface could be monotonic, however another could contain several local minima. Since the simulated annealing algorithm has the ability to stochastically "climb" out of local minima, a priori knowledge of the optimization surface is not necessary. An additional advantage of the simulated annealing algorithm is that the implementation is general enough that it could be extended to solve multiple way point problems.

II. DETAILED PROBLEM DESCRIPTION

The problem in question is to develop an optimization technique which could be run on board the space station and is capable of determining minimum fuel rendezvous trajectories. It is assumed that the rendezvous is a proximity operation implying that the mission should be completed less than one orbit.

The space station environment will include a variety of spacecraft co-orbiting in close vicinity. This new environment is described in detail in Grunwald and Ellis and summarized here [1and 6]. The multivehicle environment will include new requirements which do not exist in conventional mission scenarios. Additionally, the set of possible scenarios in a multivehicle environment is nearly unlimited. Therefore, the space station environment could create scenarios that have not been envisioned and that will have to be planned and executed on board. To help astronauts plan such missions, NASA AMES Research Center has developed an Interactive Orbital Proximity Operations Planning System, known as Navie. The optimization scheme presented here is intended to be used in conjunction with Navie.

Navie is a graphic tool which can be used to plan proximity operations. A graphic tool is especially useful for planning proximity operations because there are difficulties encountered when planning and carrying out orbital maneuvers. (Refer to Appendix B, Figure 3) The first such difficulty is the counterintuitive character of orbital motion as experienced in a relative reference frame. In this situation, orbital motion is expressed relative to the space station. Intuitively, one would assume that a thrust in a forward direction would result in straightforward motion. However, a forward thrust moves the craft upward into a higher orbit. Since objects in higher orbit move more slowly than objects in lower orbit, the craft's eventual relative motion is backward, not forward! (Refer to Appendix B, Figure 1)

A second difficulty in planning proximity operations is the manner in which orbital maneuvering control forces are applied. Instantaneous thrusts, known as burns are made. The difficulty is determining the magnitude of the burn necessary for any mission. The required burn is directly dependent on the orbital mechanics. Grunwald and Ellis develop a straightforward mathematical

approach for coping with this difficulty [1 and 6]. Their method takes advantage of "inverse dynamics" and is used in this research. The advantages of the inverse dynamics approach are that it reduces the order of the control and that it linearizes the control.

The final difficulty is the requirement for safely in the space station environment. The purposes of the requirement are numerous. Several examples of realistic requirements are as follows: preventing flight close to structures, ensuring that the plume from a burn does not harm any structure or equipment, and mandating that there is zero relative velocity between two rendezvousing spacecraft. Safety requirements are implemented as constraints in the optimization code.

All of the above complexities will make it difficult to plan trajectories on demand. Navie makes the task of planning feasible trajectories much easier. However, even with the assistance of a graphic tool such as Navie, planning optimal or near optimal trajectories is a difficult assignment. It was hypothesized that a numerical optimization scheme might be able to assist users in finding near optimal solutions. The optimization to be described in section VI is capable of finding a minimum fuel trajectory for a two burn maneuver. Not all mission scenarios can be solved with two burns due to the orbital dynamics or the constraints involved. Those scenarios are not considered here.

In order to create a useable model of the problem, some underlying assumptions have been made:

1. The station is stabilized in circular orbit with inclination 28.5 degrees with respect to the equatorial plane
2. The station is orbiting the earth at an altitude of 480 kilometers
3. Trajectories of both chase and target craft are in the orbital plane
4. The eccentricity of all trajectories is low (i.e less than 0.05)

III. ORBITAL MECHANICS

Once the initial conditions are input, it is possible to compute the required burn using a set of orbital equations. Assuming that the mission initiation is set to a relative time 0, the necessary input information is as follows:

1) Target craft : Initial position relative to the space station
Magnitude of relative motion
Direction of relative motion

2) Chase craft: Initial position relative to the space station
Magnitude of relative motion
Direction of relative motion

3) Time to rendezvous

The mathematical background is described in detail in Grunwald and Ellis.[1 and 6]. The equations that are required to compute trajectories given the input data, will be described here. The orbital equations are derived from Kepler's equations of motion. These equations cannot be used directly because a simple closed-form solution does not exist for the rendezvous problem. A common practice in celestial mechanics is to use a Fourier-Bessel series expansion. The resultant equations describing relative motion in a Local Vertical and Local Horizontal reference frame are as follows:

$$s(t) = s(t_0) + R_0 [\, \Delta n\, (t - t_0) + [2e\, (\sin M_1 - \sin M_0)\,]\,]$$

$$r(t) = r(t_0) + [ae\, (\cos M_0 - \cos M_1)]$$

where

V-bar is a section of the circular orbit which is followed by the Space Station center of mass

R-bar is the radial line moving outward from the earth center through either the spacecraft or the Space Station

s is the distance measured along the *V-bar* between the space station and the spacecraft's ***R-bar***

r is the distance of the spacecraft above the *V-bar*, measured along the *R-bar*

R_0 is the radius of the space station orbit

Δn is the difference in *mean motion* between spacecraft

t_0 and t are the *initial time* and *time after departure*

e is the *eccentricity* of the orbit

a is the semimajor axis of the orbit

M_0 and M_1 are the *initial Mean anomaly* and the *Mean anomaly at time t*

The first computations are to determine the set of orbital parameters for the target craft. These parameters completely describe the motion of the target craft. The computations are performed in the function *orbpar* of Appendix A which is the computer program implementing the optimization scheme in the C programming language. Next, the orbital parameter set is used to determine the relative position of the target craft at a specified rendezvous time. These computations are completed in the function *orbpnt* in Appendix A. At this point the relative rendezvous position is known from the previous set of computations, and the initial position of the chase craft is known from the input data.

It is necessary to compute the burn which will position the chase craft in the relative rendezvous position at the rendezvous time. Grunwald and Ellis use the term "inverse dynamics" to describe this step. [1] Those computations are executed in the function *rendezvous* in Appendix A. The output from *rendezvous* are the directional components of the chase craft motion at mission initiation and at rendezvous. The magnitude of the first burn is simply the square root of the sum of squares of the orthogonal components of motion at mission initiation. The associated units of the burn are meters per second.

It is likely that the chase and target crafts will not have the same relative motion at rendezvous. It is therefore necessary to conduct a retro-burn which has the effect of matching the relative velocities of the two spacecraft. The magnitude of that burn is computed in the function *space_calc* in Appendix A. The total fuel use is the sum of the first burn and the retro-burn.

Using the orbital equations given above, it is possible to create a plot of fuel use versus time of rendezvous. Given any time of rendezvous, it is possible to compute the fuel use using the procedure described above. Since the fuel use for a given mission is a complex trigonometric function of the input data, the resultant plot is likely to be nonlinear. It is possible for several local minima to be present in addition to the global minima. The constraints which have not yet been discussed

mathematically, can add additional boundaries to the output surface. A complete discussion on the constraint implementation is given in Section VI.

IV. CHOICE OF OPTIMIZATION TECHNIQUE

In the space station environment, many constraints will exist. In order to develop a workable prototype of a trajectory planning tool, several simplifying modifications have been made. For example, only trajectories consisting of two burns are considered and only four constraints are to be implemented. In order to appreciate the choice of optimization algorithms, it is necessary to understand the system requirements. An underlying requirement was to create the optimization scheme in such a way that it would be easily extendable to solve more complex problems in subsequent research stages.

During this prototype phase the surface to be minimized can be seen as a plot of fuel use, which is the dependant variable versus time of rendezvous. Note that a possible extension would be to allow the mission to commence at some future time. The goal of the optimization would then be to find the time of initiation and the time of rendezvous leading to the minimum fuel use. In this extended problem, the surface has 3 dimensions with fuel use plotted versus time of initiation and time of rendezvous. That surface is constrained by the fact that the time of rendezvous must be greater than time of initiation. The optimization scheme should be capable of solving either the basic or the extended problem.

The surface to be optimized is completely dependant on the initial conditions of the chase and target crafts. As those initial conditions change, so does the output surface. In fact, small changes to the input conditions can affect the output surface abruptly. Furthermore, it is likely that the surface will contain local minima. Both are consequences of the nonlinear nature of the orbital mechanics. The optimization should be able to find optimal or near optimal solutions to any problems which are described by reasonable sets of input data. The resultant optimization surfaces from such varying input data will be diverse.

It would be difficult to use deterministic descent optimization algorithms to solve this general

problem. In the presence of local minima descent algorithms will get caught in the first minima that is found. If the location of the local minima were known, intelligent starting points could be used. However, prior knowledge is not possible in this problem because of the dependence on the input conditions. One possible solution would be to use random starting locations coupled with a descent algorithm. After running the algorithm a number of times, and comparing the solutions, an optimal or near optimal solution could probabilistically be obtained.

Another possibility is to try to fit a high order polynomial to the surface, and then optimize that fitted surface. This would involve computing the fuel use for some number of points, fitting a polynomial to those points using least squares, and then optimizing the fitted surface. This approach was tried. The experimental results demonstrated that the sensitivity of the method is inadequate for this problem. Residual analysis was conducted on known surfaces in experiments composed of up to 150 test points fit to a fifth degree polynomial. The minimum point of the fitted surfaces did not consistently approximate the actual minimum.

Simulated annealing is a stochastic optimization algorithm which has been shown to be capable of solving complex problems. [2]. The theory of how simulated annealing works is described in detail in van Laarhoven. [3] The work that has been completed here is an application of that which is described there. The two main advantages that this technique offers this problem are its ability to optimize complex surfaces without requiring a priori knowledge of that surface and its ability to stochastically "climb" out of local minima . Futhermore, it has been stated that for most problems simulated annealing performs better than descent methods repeated at a number of different initial solutions [4].

V. SIMULATED ANNEALING THEORY

The simulated annealing algorithm is based on an analogy. The analogy is between the annealing of solids and the problem of solving combinatorial optimization problems [5]. To appreciate the

subtleties involved in the simulated annealing algorithm, an understanding of the analogy with physical annealing is necessary.

Annealing is a thermal process used to obtain low energy states of a solid. The solid to be annealed is put in a heat bath which is heated to a point at which the solid melts. The temperature of the heat bath is then slowly reduced. At each stage of temperature reduction, the material achieves thermal equilibrium. A material is said to be in thermal equilibrium if the distribution of its energy levels may be described by the Boltzmann distribution:

$$\text{PROB(Material is in state i with Energy } E_i \text{)} = 1/Z(T) * \text{EXP } (-E_i / (K_b * T))$$

where $Z(T)$ = Partition Function

K_b = Boltzmann Constant

If the temperature (T) starts at a high enough point, and is reduced in such a way that equilibrium is attained at each stage, the system will eventually settle into the ground state. The ground state of a system is the state with the lowest obtainable energy.

In simulated annealing, an analogy between an optimization problem and a physical many-particle system is assumed. Sets of input parameters to an optimization problem correspond to states of a physical system. The objective function of the optimization problem corresponds to the energy of a state in the physical system. Finally, an artificial control parameter is created to correspond to the temperature in physical annealing. Given the above components of the analogy, it can be seen that the goal of the simulated annealing algorithm is to find a set of input parameters which yield the lowest possible objective value.

In carrying out the simulated annealing algorithm, the first step is to "heat" the optimization. When a solid is heated to the point of melting as is done in physical annealing, states corresponding to low and high energy are possible. Simulated annealing probabilistically heats an optimization problem, so that all input states are equally likely regardless of the associated objective value. A transition from one state to another can either be accepted or rejected. When the simulated annealing algorithm begins, the probability that a transition is accepted approaches 1. The acceptance criterion for transitions is as follows:

PROB(Accept a transition from state i to state j) =

1.0	If Cost (j) <= Cost (i)
EXP (- [Cost(j) - Cost(i)] / Control Parameter)	Otherwise

Using the acceptance criterion, it can be seen that the probability of accepting an increase in objective function approaches one if the control parameter is high. Also note that the probability of acceptance approaches zero as the control parameter value tends to zero. With these concepts in mind, it is apparent that the simulated annealing algorithm proceeds by starting with a high control parameter value, i.e. temperature, and slowly decreases that value until a state of minimal cost, i.e. energy, is achieved. Mathematically, the simulated annealing algorithm is best modeled using the theory of Markov chains: each transition depends only the the outcome of the previous transition [5]. A full description of the Markovian theory applied to the simulated annealing algorithm is given in van Laarhoven and Aarts [5]. The most important result is that the set of transitions which occur at every control parameter value can effectively be modeled as a Markov chain.

An important aspect in the analogy between physical and simulated annealing is the rate at which the temperature is cooled. The timetable which determines how and when to change the temperature (or control parameter) is called a cooling schedule. Many cooling schedules have been developed and reported [3]. It has been suggested that different cooling schedules do not affect the quality of the final solution, only the computational effort involved in attaining it, provided that the annealing is conducted properly [4]. Nevertheless, there are three aspects which must be controlled if the simulated annealing algorithm is to be carried out appropriately.

1. The initial control parameter value must be sufficiently high
2. The final control parameter value must be sufficiently low
3. An equilibrium must must be obtained at each control parameter value

In this research, a geometric cooling schedule has been used. A geometric cooling schedule is a conceptually simple cooling schedule which has been shown to be effective. [2]. There are four facets to the geometric cooling schedule:

1. Initial value of the control parameter
2. Final value of the control parameter

3. Length of the Markov chain

4. Decrementing scheme for the control parameter

The first variable to be studied is the initial value for the control parameter. When considering physical annealing, it is not difficult to determine the initial temperature of the system. In the simulated scenario, the initial control parameter value must be determined experimentally. It has been reported that if 80% of all transitions are accepted initially, the control parameter is set correctly[5]. Using the acceptance criterion described previously, experiments need to be conducted to determine the control parameter value which yields a transition acceptance ratio of 0.8. The transition acceptance ratio is defined below.

$$\text{Transition Acceptance Ratio} = \frac{\text{\# Accepted Transitions}}{\text{\# Proposed Transitions}}$$

The next aspect to consider is the stop criterion. There are several manners in which this can be implemented. The two most common are either to fix the number of values the control parameter is to take on, or by terminating the algorithm when the last configurations of consecutive Markov chains are identical for a given number of chains[3]. In this research the first approach is used. The final control parameter value must approach 0. Practically, the final value will be one which yields acceptably accurate solutions within a reasonable time frame.

The third variable is one which determines when equilibrium has been reached at each temperature. Mathematically, this corresponds to the length of the Markov chains. In practice, it is difficult to determine if an equilibrium state has been achieved. However, several easily implementable rules have been proposed. The one that is used here is an intuitive argument that a minimum number of transitions should be required at each control parameter value. A consequence of this rule is that the minimum Markov chain length increases as the control parameter decreases. This is due to the fact that the probability of accepting a transition diminishes as the control parameter is reduced.

The final consideration is the procedure by which the control parameter is reduced. The fundamental concept behind the process is to choose a decrementing value such that small Markov chain lengths are adequate to reestablish equilibrium in the system. The rule that is used in this research is :

Control Parameter Value at step k+1 = Control Parameter at step k * Decrement Value

This rule was originally proposed by Kirkpatrick[5]. Clearly, the decrement value is bounded on [0,1] . Values in the range [0.8, 0.99] are ordinarily used.The method of simulated annealing which has been implemented is as follows:

Tf= Final Control Parameter Value

Ti= Initial Control Parameter Value

D= Decrementing Value

L= Number of Accepted Transitions before decrementing control parameter

t= Current control parameter value

l= Number of Accepted Transitions at a given control parameter value

```
Experimentally determine and set Tf,Ti,D,L
    l=0;
    t=Ti;
Randomly generate a feasible starting point and compute the objective value.
present point = starting point
While ( t > Tf)
    {
      While ( l < L)
        {
          Move to a randomly generated neighboring point and compute the objective value
          If the trial point has lower objective value than present. point,
                        a) accept it as the present point
                        b) l = l+1
          Else if exp(-Change in objective value / t) > random [0,1]
                        a) accept trial point as the present point
                        b) l=l+1
        }
        l=0;
        t= t * D;
    }
```

VI. SIMULATED ANNEALING APPLIED TO THE OPTIMAL TRAJECTORY PROBLEM

The pseudo-code in the previous section fundamentally describes the application of simulated annealing that is used here. Note that the number of transitions at each control parameter value is not constant. Rather, the number of accepted transitions at each control parameter value is kept constant. Therefore, as the control parameter drops, causing the probability of accepting a rise in objective function to decrease, the number of transitions in successive Markov chains is likely to increase. When the Markov chain length is variable, the algorithm is inhomogeneous. Convergence of such an inhomogeneous algorithm is discussed in van Laarhoven and Aarts. [5] It is theoretically possible for the simulated annealing algorithm to terminate arbitrarily close to the minimum of a surface. To obtain the exact solution with probability one, an infinite number of transitions is required. In practice it is necessary to implement an approximation algorithm. The purpose of approximation algorithms is to find near optimal solutions within a reasonable amount of computation time.

It can be shown that the speed of convergence of the simulated annealing algorithm depends on the complexity of the surface to be optimized. The surfaces to be studied here are potentially complex because of the dependance on the orbital mechanics and the existence of constraints on the objective function. A barrier method is implemented to ensure that none of the constraints are violated in the predicted optimal solution. If any of the constraints are violated, the objective function is penalized by a constant amount. The penalty is large enough to force the annealing to consider alternative trajectories. In cases where descent algorithms are used for optimization, extreme care must be taken when using barrier methods. In those algorithms, inverting ill-conditioned matrices is a usual concern. Since simulated annealing does not do any matrix operations, thisl concern about using a barrier method is unwarranted.

The most time consuming part of the optimization procedure is to find an appropriate cooling schedule. In that respect extensive experimentation has been completed in this thesis . The goal of those experiments was to explore a range of possible scenarios which could occur once the space station is in place. It has been determined that the following parameter values seem to be fairly robust with respect to the surfaces that were investigated:

Initial Control Parameter value, Ti:	35.0
Decrementing Value, D:	0.95
Number of accepted transitions at a given control parameter, L:	8

However, the required final control parameter value varies among surfaces. More complex surfaces generally demand a lower final control parameter. The disadvantage of allowing the annealing to run to a lower control parameter is that the computations are more intensive. Therefore, it is desirable to use the highest final control parameter value which consistently yields a near optimal solution. That value varies among surfaces. From a practical standpoint, it would be appealing if there were a single cooling schedule which could solve all possible mission scenarios satisfactorily and efficiently. That matter is currently under investigation.

It could be argued that other optimization techniques are capable of solving the two burn problem more efficiently. That is true, especially when the mission initiation is set to a relative time 0. In this 2 dimensional case, a simple grid search may be most effective. However, a distinct advantage of the simulated annealing algorithm is that it is capable of handling extensions to the problem with little change to the optimization code. The ability to optimize more complex problems will be of paramount importance if the ultimate goal of achieving a workable trajectory planning tool is to be realized. The most likely extension to this problem is the addition of way-points. The addition of a third way point to the problem increases the complexity of the problem immensely. The location of the third way point is variable in time and in space, leading to a much larger solution space. Possible methods of implementing a multiple way point optimization are discussed in the final section.

VII. TEST CASES

In order to illustrate the simulated annealing algorithm several test cases have been developed. In all these cases, the simulated annealing optimization technique was able to find near optimal solutions. Each test case illustrates a different aspect of optimization via simulated annealing. The first shows the ability of the algorithm to find a global minimum in a case where local minima exist. Another example shows the importance of a suitable cooling schedule. The final examples show that the simulated annealing algorithm is capable of finding minimal cost solutions in cases where the

optimization surface is difficult to solve.

The purpose of the first example is twofold. First, it illustrates the ability of the simulated annealing algorithm to find a global minimum in a case where several minima exist. Secondly, it demonstrates how to develop a workable cooling schedule. The function to be examined is the following polynomial over the range [0,17]:

$$y = 0.028 * x^5 - 0.90 * x^4 + 8 * x^3 - 10 * x^2 - 50 * x + 300$$

A graph of that function is given in Appendix B, Figure 2. It can be seen that two minima exist in the specified range In order to find the global minima using the simulated annealing algorithm, it is necessary to create a usable cooling schedule. The first step is to determine the initial value of the control parameter. That is done by setting the transition acceptance ratio which is defined in section V equal to 0.8. Equivalently, the following formula may be used solving for the denominator in the exponent:

$$0.8 = \text{EXP}\ (\ -\ \text{Max change in objective function} / \text{Control Parameter Value})$$

The maximum change in objective function may be determined from a graph such as Appendix B, Figure 2. Note that it is necessary to have defined a transition range in order to determine the maximum change in the objective function. The transition range is simply the maximum allowed distance between trial points. In the current example, the transition neighborhood is defined to be +/- 1.5 units. The maximal change in objective value is therefore found to be 365.37. Using the formula above, the initial control parameter value is computed to be 1637.38. The initial control parameter is set to 1650.

The final control parameter is set to 0.01. This was determined experimentally through trial and error. The other two parameters, chain length and decrement value, were also determined experimentally. The following table gives some of the results which led to the choice of the parameter set.

Czero	Cfinal	Chain Length	Decrement Value	# Runs	# Times Correct
1650	0.01	12	0.95	5	3
1650	0.01	12	0.975	5	2
1650	0.01	18	0.95	5	3
1650	0.01	18	0.975	5	3

1650	0.01	25	0.95	10	4
1650	0.01	25	0.975	10	7
1650	0.01	50	0.95	5	4
1650	0.01	50	0.975	5	4
1650	0.01	75	0.95	5	2
1650	0.01	75	0.975	10	10

Using a Silicon Graphics 4D-120 the computations detailed above were completed. The last row in the table gives suitable parameter values for the annealing procedure. On average, those computations took nearly 200 cpu seconds. Note that this particular problem could easily have been solved more efficiently using other optimization techniques. However, the procedure which was used to determine the cooling schedule here is easily generalizable to more difficult problems.

An interesting alternative to solving the original problem is to solve a scaled version of the original problem. Scaling is a common technique when using descent algorithms. By scaling the original problem, the simulated annealing algorithm will be able to start at a lower initial control parameter value. This fact follows from the fact that the maximum change in cost has been scaled. The result is a decrease in computation time. The original problem was arbitrarily scaled by a factor of .01. The new polynomial to be optimized is the following:

$$y' = 0.00028 * x^5 - 0.0090 * x^4 + 0.08 * x^3 - 0.1 * x^2 - 0.5 * x + 3.0$$

The following table describes the experiments which were conducted:

Czero	Cfinal	Chain Length	Decrement Value	# Runs	# Times Correct
20.0	0.01	75	0.95	5	3
20.0	0.01	75	0.975	5	4
50.0	0.01	50	0.975	20	18
50.0	0.01	60	0.975	10	6
50.0	0.01	65	0.975	10	6
50.0	0.01	75	0.95	5	2
50.0	0.01	75	0.975	20	20

It is easily seen that the most suitable parameter set is (50.0, 0.01, 75, 0.975). The average time to solve the scaled problem using the above parameter set was 19.1 cpu seconds. The sensitivity of the solution to the minima will be affected by scaling. However, if the problem is such that computation time is an important factor, and some sensitivity about the mimima can be sacrificed, minimizing a scaled problem may be a viable solution.

EXAMPLE 1:

It was claimed that several criterion must be met if the simulated annealing algorithm is to find the optimum of a given surface. Those criterion are that the initial control parameter must be high enough, that the final control parameter value must be low enough, and that equilibrium must be obtained at each control parameter value. This example demonstrates the validity of those claims. All positions, directions and magnitudes are relative to the Space Station.

The input information for this test case is as follows:

Relative initial location of Chase craft: (x,y) in meters	(-200.0, 400.0)
Relative magnitude of chaser initial motion: (meters/second)	0.0
Relative direction of chaser initial motion: (radians)	0.0
Relative initial location of Target craft:	(4.5,140.0)
Relative magnitude of target initial motion: (meters/second)	0.0
Relative direction of target initial motion: (radians)	5.06
Allowed departure angle (degrees)	160.0
Allowed arrival angle (degrees)	135.0
Constrained spatial coordinates (x range),(y range)	(-80,80),(-260,160)
Maximum rendezvous time (minutes)	90.0
Maximum transition neighborhood (minutes)	+/- 3

The x axis lies in the same direction as the tangent to the space station's velocity. A positive value corresponds to being in front of the station. The y axis is the direction of the height of the orbit. A positive y value corresponds to an orbit below the station. In this particular case, the target starts in front of (4.5 meters) and below (140 meters) the station, and is released at an angle of 5.06 radians (289.92 degrees) relative to the positive x-axis at the space station, with zero relative motion. (Angles

are measured counter clockwise from that axis) The orbital mechanics force the target to go below and in front of the station. The chase craft starts behind (200 metres) and below (400 metres) the station. The mission initiates at a relative time 0. Navie's graphic representation of a 20 minute rendezvous mission is given in Appendix B, Figure 3.

The goal of the optimization is to find the rendezvous time which yields the lowest fuel use and concurrently does not violate any of the constraints. A plot of the surface to be minimized is given in Appendix B, Figure 4. The area near the minimum in Appendix B, Figure 4 is shown in detail in Appendix B, Figure 5. From those figures it can be seen that the minimum fuel use is 0.615 m/s. The goal of the optimization is to reliably find the minimum to within .001 m/s. The following table shows how different cooling schedules affect the outcome of the optimization.

Expt #	Initial Control	Final Control	#Accepts	Decrement	Final fuel	Time of Rendezvous	CPU Time
1	20.0	.004	8	.95	.6196 m/s	34.62 min	41.41 sec
					.6295	38.83	77.07
					.6155	30.74	61.54
					.6157	30.33	96.77
2	20.0	.001	8	.95	.6168	32.91	106.78
					.6161	32.16	114.41
					.6157	30.34	118.98
3	35.0	.008	8	.95	.6156	31.33	67.63
					.6169	32.92	71.34
					.6171	33.07	72.83
					.6187	34.15	79.97
4	35.0	.004	8	.95	.6155	30.31	50.5
					.6157	30.82	99.99
					.6186	34.14	112.9
					.6160	29.88	106.8
5	35.0	.004	5	.95	.6156	31.17	33.77
					.6158	31.77	22.05
					.6156	31.41	33.54

					.6162	32.32	43.07
6	35.0	.004	3	.95	.6196	34.75	9.78
					.6165	32.63	21.06
					.6155	30.97	8.45
					.6156	31.36	11.94
7	35.0	.001	8	.95	.6157	30.19	159.46
					.6157	30.23	139.49
					.6168	29.88	89.10
					.6165	32.58	97.54

It can be seen from the above data that indeed several criteria must be met in order for the simulated annealing to consistently find the required minimum. Unfortunately, there does not seem to be any well founded theory which explains the relationship between the nature of surfaces of interest and the the required cooling schedule to optimize the surface. Numerical experiments such as those shown above must be completed in order to determine what parameter values are suitable for particular optimization surfaces.

EXAMPLE 2:

The following two examples assume that a reasonable cooling schedule has been developed. Their purpose is to demonstrate that the simulated annealing is capable of finding near optimal solutions even in cases where the surface to be optimized is degenerate. The input data is the following:

Relative initial location of Chase craft: (x,y) in meters	(-600.0, 800.0)
Relative magnitude of chaser initial motion: (meters/second)	0.0
Relative direction of chaser initial motion: (radians)	0.0
Relative initial location of Target craft:	(8.5,-145.0)
Relative magnitude of target initial motion: (meters/second)	0.25
Relative direction of target initial motion: (radians)	3.84
Allowed departure angle (degrees)	160.0
Allowed arrival angle (degrees)	30.0
Constrained spatial coordinates (x range),(y range)	(-80,80),(-260,160)
Maximum rendezvous time (minutes)	90.0

Maximum transition neighborhood (minutes) +/- 3

A graphic simulation of this input scenario is given in Appendix B, Figure 6. A plot of fuel use versus time of rendezvous is given in Appendix B, Figure 7. It can be seen that the minimum area is a relatively large flat section in the neighborhood of 38 to 61 minutes. A plot of fuel use versus time in the region of interest is given in Figure 8. The absolute minimum for this problem occurs next to a barrier which represents a violated constraint. The cooling schedule used to optimize the surface specified here was (35.0, 0.008, 8, 0.95). The cooling schedule was determined through a series of experiments similar to those in the previous example. The following are the simulation results:

Initial Control	Final Control	#Accepts	Decrement	Final fuel	Time of Rendezvous	CPU Time
35.0	0.008	8	0.95	2.12 m/s	39.81 min	86.11sec
				2.11	39.15	97.31
				2.12	39.66	83.01
				2.12	39.61	68.47

Note that the "optimal" solutions which are obtained above are not the exactly equal to the minimal point on the graph. It is possible to get arbitrarily close to the absolute minimum of a given problem. However, as the requirement to do so increases, so does the computation time required to achieve the goal. For the application at hand, the required sensitivity to the absolute minimum is in the neighborhood of 0.02 m/s. Since the annealing optimization will have to be run on board of the space station, it is necessary that solutions be obtained as quickly as possible. The example above demonstrates that a near optimal solution is obtainable in a realistic amount of computation time.

EXAMPLE 3:

The following example is similar to the previous one, except the rendezvous window is much smaller and the optimization surface is less well behaved. The input conditions are the following.

Relative initial location of Chase craft: (x,y) in meters (70.0, -40.0)

Relative magnitude of chaser initial motion: (meters/second)	0.0
Relative direction of chaser initial motion: (radians)	0.0
Relative initial location of Target craft:	(8.5,-145.0)
Relative magnitude of target initial motion: (meters/second)	0.25
Relative direction of target initial motion: (radians)	3.84
Allowed departure angle (degrees)	160.0
Allowed arrival angle (degrees)	40.0
Constrained spatial coordinates (x range),(y range)	(-80,80),(-260,160)
Maximum rendezvous time (minutes)	90.0
Maximum transition neighborhood (minutes)	+/- 3

Refer to the graphic simulation in Appendix B, Figure 9. The data is such that rendezvous cannot occur before 58.4 minutes because of the spatial constraint, and cannot occur after 59.3 minutes because of the angular arrival constraint. There is approximately a one minute window where a trajectory would not be penalized. (refer to Appendix B, Figures 10 and 11) The fuel use within that window remains fairly constant around 0.21 m/s. The fact that the optimal area is relatively flat with respect to fuel use is not uncommon. Therefore, an additional feature to check for a flat optimal area has been built into the optimization. The result is that the program will find a near optimal trajectory, and then determine if alternative trajectories will yield similar results. If so, the optimal range is output. The sensitivity of that range is variable .

To solve this particular problem, the following cooling schedule is used:

Initial Control Parameter, Ti:	35.0
Final Control Parameter, Tf:	0.90
Control Parameter Decrementing Value:	0.95
Number of accepted transitions per control parameter:	8

The spike in the optimization surface is purely a function of the method of constraint implementation. The only disadvantage to using a barrier method with the simulated annealing optimization is that finding an appropriate cooling schedule can become unusually difficult. In addition to the usual problems associated with finding a suitable cooling schedule, special attention must be given to the final control parameter value. Ordinarily, if the final value for the control parameter is low enough,

the optimization will terminate in an acceptable state. However, when a barrier exists on either side of the minimum, as is the case in this example, the final control parameter is a concern.

Recall that as the control parameter decreases, so does the probability of accepting an increase in objective value. When constraints are implemented via barrier methods, the resultant surface is likely to have "spikes". A problem occurs when the control parameter value is low, and the system is in the spiked area of the surface. In order to terminate, the algorithm must accept a given number of transitions before decrementing the control parameter. Clearly, the probability of accepting a transition is low when the system is in the spiked area of the surface. The resultant computations can become unreasonable. The solution is to find the highest final control parameter value which finds a near optimal solution consistently.

The following are simulation results:

Initial Control	Final Control	#Accepts	Decrement	Final fuel	Time of Rendezvous	CPU Time
35.0	0.90	8	0.975	0.21 m/s	59.12 min	16.26 sec
				0.21	58.39	19.88
				0.21	58.87	30.92
				0.21	58.55	26.14

Once again it can be seen that the simulated annealing algorithm was able to find near optimal solutions within a reasonable amount of time.

VIII. EXPERIMENTAL

To demonstrate that the simulated annealing algorithm is capable of solving the general two burn rendezvous problem in the neighborhood of the space station, a systematic series of experiments was prepared. The experimental sequence simulates a situation where an object is released from the

neighborhood of the space station with a given magnitude and direction of motion. Eight positions in the space station neighborhood are considered. They correspond to the four corners of the spatially constrained area and the four mid-points of the area boundary. At each position, an object is released at two different magnitudes and at eight different angles. The chase craft starting position is a constant in all of the experiments. The chase craft is positioned in the same orbit as the space station, but 60 meters ahead of it. There are a total of 128 experimental points.

The results of the experiments will show that the final configurations of the optimization approximates the true optimal solution for each input case, and does so in a reasonable amount of time. Because missions are limited to 2 burns, it is possible that no feasible trajectory will exist for a given input set. Recall that a feasible trajectory does not violate any of the constraints. In cases where no feasible solution exists, the optimization reports that fact and terminates. The most common reason for infeasibility is that the chase craft must fly through the spatially constrained area in order to rendezvous with the target craft.

The following cooling schedule was used for all experiments:

Initial Control Parameter	35.0
Final Control Parameter	0.01
Control Parameter Decrementing Value	0.95
Number of accepted transitions per control parameter	8

The experiment was conducted on a Silicon Graphics 4D-120, which runs at 20 MIPS and 2 Mega Flops. The following table describes the results of the experiments.

Initial Target Location		Magnitude of	Direction of	Feasible	Final	Best	CPU time
x(meters)	y(meters)	Motion(m/s)	Motion (rad)	(yes/no)	Solution	Solution	(secs)
-80.00	260.00	0.05	all	no			
		0.30	all	no			
-80.00	-160.00	0.05	0.000	yes	0.55	0.54	38.50
			1.571	yes	0.52	0.51	40.51
			3.142	yes	0.45	0.44	47.48
			4.712	yes	0.48	0.47	37.27
			0.785	yes	0.54	0.54	37.61
			2.356	yes	0.47	0.47	32.76
			3.927	yes	0.45	0.44	55.33
			5.498	yes	0.51	0.51	35.02

		0.30	0.000	yes	0.78	0.78	30.37
			1.571	yes	0.70	0.68	59.03
			3.142	no			
			4.712	yes			> 120
			0.785	yes	0.78	0.77	49.92
			2.356	yes	0.50	0.50	52.15
			3.927	no			
			5.498	yes	0.65	0.63	27.87
80.00	260.00	0.05	all	no			
		0.30	all	no			
80.00	-160.00	0.05	0.000	yes	0.49	0.47	46.77
			1.571	yes	0.45	0.45	49.83
			3.142	yes	0.39	0.39	26.34
			4.712	yes	0.42	0.41	42.41
			0.785	yes	0.48	0.48	44.94
			2.356	yes	0.42	0.42	46.38
			3.927	yes	0.39	0.38	56.09
			5.498	yes	0.45	0.45	49.94
		0.30	0.000	yes	0.72	0.71	53.90
			1.571	yes	0.61	0.59	50.71
			3.142	no			
			4.712	no			
			0.785	yes	0.72	0.71	48.53
			2.356	yes	0.40	0.40	69.49
			3.927	no			
			5.498	no			
0.00	-160.00	0.05	0.000	yes	0.51	0.51	34.66
			1.571	yes	0.51	0.48	31.54
			3.142	yes	0.42	0.41	44.22
			4.712	yes	0.45	0.45	38.59
			0.785	yes	0.51	0.51	41.20
			2.356	yes	0.44	0.44	45.18
			3.927	yes	0.42	0.42	31.07
			5.498	yes	0.49	0.49	36.94
		0.30	0.000	yes	0.75	0.74	25.08
			1.571	yes	0.63	0.63	51.96
			3.142	no			
			4.712	no			
			0.785	yes	0.74	0.74	49.56
			2.356	yes	0.46	0.45	50.52
			3.927	no			
			5.498	yes	0.62	0.61	27.32
80.00	0.00	0.05	all	no			
		0.30	all	no			
0.00	260.00	0.05	all	no			
		0.30	all	no			
-80.00	0.00	0.05	all	no			
-80.00	0.00	0.30	0.000	yes	0.36	0.36	25.22
-80.00	0.00	0.30	0.785	yes	0.47	0.45	20.14
		0.30	all others	no			

An entry of "no" in the Feasible column above is interpreted to mean that no feasible solution exists for the corresponding input set. The best solution that is reported in each feasible case is the lowest objective value that was found during the course of the optimization. The probability that the lowest objective value tends to the true minimal solution approaches one as the number of sample

points increases.

It can be seen that only 41 of 128 experiments produce feasible results. Considering those cases where feasible solutions exist, the simulated annealing algorithm performs favorably in finding near optimal solutions. In 39 of the 41 cases, the simulated annealing algorithm terminated at a near optimal solution in less than one minute of cpu time. In all but one case the algorithm's final solution was within .02 m/s of the global minimum. In that case the algorithm terminated because the time required to solve the optimization was greater than 2 minutes. A frequency distribution of computation times for input sets producing feasible results is given in Appendix B, Figure 12.

IX. DISCUSSION

In the previous section it was demonstrated that the simulated annealing algorithm is capable of finding near optimal solutions to a variety of realistic problems. Futhermore, the computation time involved in finding those optima is reasonable. It is important to recall that the ultimate goal is to develop a tool which can aid astronauts in planning trajectories on board of the space station. The optimization procedure which has been developed here is only one step in developing such a tool.

Realistic trajectories will most likely contain more than two burns. It is for that reason that the simulated annealing algorithm seems more appealing than other algorithms, particularly grid searches. With the addition of a third way point in a given trajectory, the solution space increases from two dimensions to a five dimensional surface. The additional degrees of freedom are the x and y coordinates of the new burn, and the time of the new burn. Grid searching over multidimensional solution spaces can easily become impractical .

The major advantages that the simulated annealing algorithm displays for this application are its ability to stochastically "climb" out of local minima and its' inherent ability to be extended to more complex problems. There are problems that will be encountered when implementing the simulated annealing algorithm. Perhaps the biggest drawback to this algorithm is that it is relatively difficult to find an appropriate cooling schedule for a given problem Although several easily implementable

cooling schedules have been developed and reported, there is a lack of theory relating surface characteristics to convergence of the algorithm. While it is true that experience allows a user to "guess" reasonable cooling schedules, it appears that the only viable method for developing cooling schedules is through a series of trial and error experiments.

X. FUTURE DIRECTIONS AND CONCLUSIONS

The are two directions of research that stem directly from this work. The first is to investigate the possibility of developing a general cooling schedule which is capable of finding near optimal solutions to problems which are likely to be encountered. One possible approach to this problem is to classify the input state as one of several types. Once this is done, it may be possible to develop general cooling schedules based on the "type" of problem. The method of investigation may be seen as a statistical design problem, where the four parameters are seen as variables which can be set at a variety of levels. A fractional factorial design or response surface design may be appropriate.

The second direction of research is aimed at solving the multiple burn problem. The fact that the multiple burn problem is much more complex than the two burn problem was introduced in the previous section. A possibility is to introduce a knowledge base which contains "expert" information on how trajectories should be planned. With the introduction of such a system, the third burn could be expertly placed in a neighborhood in time and space. At that point, the optimization may be able to find minimal solutions by exploring trajectories which are perturbations of the expert starting point. There are two potential drawbacks to this solution. The first is that computationally this approach may be infeasible. The second is that there may be no true experts which could be used as a source of information. Nevertheless, the idea of using a knowledge base to direct an optimization procedure may be a topic worthy of investigation.

XI. ACKNOWLEDGEMENTS

This research was conducted as part of the cooperative agreement NCC 2-86 between NASA Ames Research Center and U.C. Berkeley. Special thanks is given to Professor L. W. Stark, the principle investigator of this agreement.

XII. BIBLIOGRAPHY

[1] A.J. Grunwald and S.R. Ellis, Interactive Orbital Proximity Operations Planning System, *NASA Technical Paper* 2839, November, 1988.

[2] S. Kirkpatrick, C.D. Gelatt, Jr., and M. Vecchi, Optimization by Simulated Annealing, *Science*, vol. 220, pp671-680, May 1983.

[3] P.J.M. van Laarhoven, *Theoretical and computational aspects of simulated annealing*, Stichting Mathematisch Centrum, Amsterdam, 1988.

[4] E.H.L. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines, A stochastic approach to combinatorial optimization and neural computing*, Wiley-Interscience, 1989.

[5] P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Company, 1987.

[6] S.R. Ellis and A.J. Grunwald, The Dynamics of Orbital Maneuvering: Design and Evaluation of A Visual Display Aid for Human Controllers, *Proceedings of the AGARD SMP Symposium on Space Vehicle Flight Mechanics*, pp. 29-1 to 29-13, Luxemburg, 1989.

APPENDIX A

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sgimath.h>
#include <sys/types.h>
#include <sys/times.h>
#include <sys/param.h>  /* for HZ */

#define drand48() (lrand48() / pow(2,31))

#define TIME0     times(&t_timbuf);t_ltime =
                t_timbuf.tms_utime;
#define TIME1(msg) times(&t_timbuf);
              printf(" msg : %5.2f sec\n", \
        (double)(t_timbuf.tms_utime - t_ltime) / HZ );

struct tms t_timbuf;
time_t t_ltime;

struct position
{
 double xpost;
 double ypost;
 double delvt;
 double alpt;
};

struct burninfo
{
 double rodot;
 double sodot;
 double rfdot;
 double sfdot;
};
```

```
struct orbset
{
 double csam1;
 double snam1;
 double dangrt;
 double angrt;
 double T1;
 double am1;
 double a;
 double ecc;
};


/* FUNCTION TO READ THE NEEDED INFORMATION FOR THE TARGET CRAFT
  FROM A FILE SPECIFIED BY THE USER          */

  struct position *rdfile(infile)
  char infile[15];

  {
  FILE  *in;
  struct position *indat;

  indat = (struct position *) malloc (sizeof(struct position));
  if( indat == (struct position *) 0)
    {
       printf("malloc failed in rdfile\n");
       exit(1);
    }

  in = fopen(infile, "r");
  if( in == NULL)
    {
    printf("file not open\n");
    exit(1);
    }
  else
    {
       fscanf(in, "%lf %lf %lf %lf", &indat->xpost, &indat->ypost,
         &indat->delvt, &indat->alpt);
       fclose(in);

       return(indat);
    }
  }
```

```
/* FUNCTION TO COMPUTE THE ORBITAL SET FROM THE POSITION AND VELOCITY
   RELATIVE TO THE SPACE STATION.  THE INPUT ARE THE INITIAL LOCATION
   MAGNITUDE AND DIRECTION OF THE CRAFT AT TIME 0.  THESE ARE PASSED
   IN THE STRUCTURE INDATA AND ARE MANIPULATED AS ININFO. THE RESULTS
   ARE STORED IN AN ORBSET CALLED SET.                    */

struct orbset *orbpar(ininfo, time)
struct position *ininfo;
double time;

{
 struct orbset *set;
 double faco;
 double Ro;
 double angrto;
 double fac;
 double R1A;
 double R1Ro;
 double E1;
 double esne1;
 double ecse1;
 double ecc2;
 double dvo;
 double snalp;
 double sq;
 double dvoca;
 double dvo2;
 double drro;
 double drro2;
 double csalp;
 double vo;
 double vo2;

 static double alt = 480.0;
 static double RE = 6.378140e6;
 static double GM = 3.986006e14;

 set = (struct orbset *) malloc (sizeof(struct orbset));
 if( set == (struct orbset *) 0 )
  {
     printf("malloc failed in orbpar\n");
     exit(1);
  }

 Ro = RE + alt*1000.0;
 vo2 = GM / Ro;
```

```
vo = _lsqrt(vo2);
angrto = vo / Ro;

snalp = _lsin(ininfo->alpt);
csalp = _lcos(ininfo->alpt);

drro = -1.0 * ininfo->ypost / Ro;
drro2 = drro * drro;
R1Ro = 1.0 + drro;
dvo = ininfo->delvt / vo;
dvo2 = dvo * dvo;
dvoca = dvo * csalp;

ecse1 = 2.0 * dvoca + dvo2 + drro * (3.0 + 4.0 * dvoca + dvo2) +
                drro2 * (3.0 + 2.0 * dvoca) + drro2 * drro;

R1A = 1.0 - ecse1;
sq = _lsqrt(R1Ro * R1A);
esne1 = dvo * snalp * sq;
ecc2 = ecse1 * ecse1 + esne1 * esne1;

E1 = 0.0;
if ( _lfabs(esne1) >= 1e-10 || _lfabs(ecse1) >= 1e-10)
   E1 = _latan2(esne1, ecse1);

set->ecc = _lsqrt(ecc2);
set->am1 = E1 - esne1;

faco = R1A / R1Ro;
fac = _lpow( faco, 1.5);
set->angrt = angrto * fac;

set->T1 = set->am1 / set->angrt;
set->T1 = set->T1 + time * 60.0;

set->am1 = set->T1 * set->angrt;
set->snam1 = _lsin(set->am1);
set->csam1 = _lcos(set->am1);

set->dangrt = -1.5 * set->angrt * (1.0 - faco);
set->a = Ro / faco;
return(set);
}

/* FUNCTION TO COMPUTE THE POSITION AND RELATIVE VELOCITY ON AN
  ORBITAL PATH FOR WHICH THE ORBITAL SET PARAMETERS ARE GIVEN */
```

```
struct position *orbpnt(set, xposo, yposo, time)
struct orbset *set;
double xposo;
double yposo;
double time;

{
 struct position *newpos;
 double drdot;
 double deldot;
 double Ro;
 double dr;
 double del;
 double t2;
 double snam2;
 double csam2;

 static double RE = 6.378140e6;
 static double alt = 480.0;

 newpos = (struct position *) malloc (sizeof(struct position));
 if( newpos == (struct position *) 0)
       {
        printf("malloc failed in orbpnt\n");
        exit(1);
       }

 Ro = RE + alt*1000.0;
 t2 = time * 60.0 + set->T1;

 if (time == 0)
     { newpos->xpost = xposo;
      newpos->ypost = yposo;
      return(newpos);
     }

 else
     {
     snam2 = _lsin(set->angrt * t2);
     csam2 = _lcos(set->angrt * t2);

     dr = set->ecc * (set->csam1 - csam2) * set->a;
     del = set->dangrt * time * 60.0 + 2.0 *
     set->ecc * (snam2 - set->snam1);
```

```
        newpos->xpost = xposo + del * Ro;
        newpos->ypost = yposo - dr;

        drdot = set->ecc * set->angrt * snam2 * set->a;
        deldot = set->dangrt + 2.0 * set->ecc * set->angrt * csam2;

        newpos->delvt = _lsqrt(drdot * drdot + deldot * deldot * Ro * Ro);
        newpos->alpt = _latan2(drdot, deldot * Ro);

    return(newpos);
    }
   }
```

```
/* FUNCTION TO CALCULATE THE BURN REQUIRED TO COMPLETE THE
RENDEZVOUS AND THE VELOCITY AT TERMINATION. THE RESULTS ARE
RETURNED IN THE FORM OF A STRUCTURE NAMED UPDATE. THE NECESSARY
INITIAL BURN IS EXPRESSED AS RODOT AND SODOT, AND THE RESULTING FINAL
VELOCITY IS RFDOT AND SFDOT    */

   struct burninfo *rendezvous (szero, rzero, sf, rf, time, n)
   double rzero;
   double szero;
   double rf;
   double sf;
   double time;
   double n;

   {
    struct burninfo *update;
    double a11;
    double a12;
    double a22;
    double delta;
    double b11;
    double b21;
    double b1;
    double b2;

    update = (struct burninfo *) malloc (sizeof(struct burninfo));
    if ( update == (struct burninfo *) 0)
          {
           printf("malloc failed in rendezvoux\n");
           exit(1);
          }
```

```
a11 = (1.0/n) * _lsin(n * time * 60.0);
a12 = (2.0/n) * (1.0 - _lcos(n * time * 60.0));
a22 = 4.0 * a11 - (3.0 * time * 60.0);

b11 = 4.0 - 3.0 * _lcos(n * time * 60.0);
b21 = 6.0 * (_lsin(n * time * 60.0) - n * time * 60.0);
b1 = rf - b11 * rzero;
b2 = sf - b21 * rzero - szero;

delta = a11 * a22 + a12 * a12;

update->rodot = (a22 * b1 - a12 * b2) / delta;
update->sodot = (a12 * b1 + a11 * b2) / delta;

update->rfdot = _lcos(n * time * 60) * update->rodot +
                    2.0 * _lsin(n * time * 60) * update->sodot +
                    3.0 * n * _lsin(n * time * 60) * rzero;
update->sfdot = -2.0 * _lsin(n * time * 60.0) * update->rodot +
                    (4.0 * _lcos(n * time * 60.0) -3.0) * update->sodot +
                     6.0 * n * (_lcos(n * time * 60.0) - 1.0) * rzero;

return(update);
}


/* The following is the implementation of the spatial constraint */
    int spatial(pen,chasein,outinfo,tt)
    int pen;
    struct position *chasein;
    struct burninfo *outinfo;
    double tt;

    {
     float testime;
     int spatialpen;
     int startin;
     struct position *chaseout;
     struct orbset *testset;
     struct position *chasetest;
     int startcheck;
     int nowcheck;


     testime = 0;
     spatialpen =0;
     startin = 0;
     chaseout = (struct position *) malloc (sizeof(struct position));
```

```
chaseout-> xpost = chasein->xpost;
chaseout-> ypost = chasein->ypost;
if ( chasein->xpost == 0. && chasein->ypost == 0.)
  {
  printf("CHASE CRAFT CANNOT START FROM THE STATION CENTER\n");
  exit(3);
  }
chaseout-> delvt = sqrt(outinfo->rodot * outinfo->rodot +
        outinfo->sodot * outinfo->sodot);
chaseout-> alpt = _latan2(outinfo->rodot, outinfo->sodot);

testset= orbpar(chaseout,testime);

if( _lfabs(chaseout->xpost) < 80. && chaseout->ypost <280.
  && chaseout->ypost > -160. )
  {

   if ( chaseout->xpost >= 0 )
     {
      if ( chaseout->ypost > 0 )
          startcheck = 1;
      else
          startcheck = 2;
     }
    else
     {
      if ( chaseout->ypost > 0 )
          startcheck = 3;
      else
          startcheck = 4;
     }
  }

while ((spatialpen == 0 || startin == 1) && testime <= tt)
  {
    chasetest = orbpnt(testset, chasein->xpost,
     chasein->ypost, testime);
    if ( _lfabs(chasetest->xpost) < 80. &&
     chasetest->ypost <280. && chasetest->ypost > -160.)
      {
       if ( chasetest->xpost >= 0 )
         {
          if ( chasetest->ypost > 0 )
            nowcheck = 1;
          else
```

```
                nowcheck = 2;
              }
            else
              {
                if ( chasetest->ypost > 0 )
                  nowcheck = 3;
                else
                  nowcheck = 4;
              }
            spatialpen = 5;
          }
        else
          spatialpen = 0;

        if (testime == 0  && spatialpen == 5.)
          startin = 1;

        if (startin == 1 && (startcheck != nowcheck) )
          {
            spatialpen = 5;
            startin = 0;
          }
        if (spatialpen == 0. && startin == 1 )
          startin = 0;

        testime += .5;
      }

    return(spatialpen);
  }

double space_calc(newset,newsetc,targin,chasein,temptt,
  chasposi,targpost,outinfo,penalty,renapert,renideal,maxinitburn)

  struct orbset *newset;
  struct orbset *newsetc;
  struct position *targin;
  struct position *chasein;
  double temptt;
  struct position *chasposi;
  struct position *targpost;
  struct burninfo **outinfo;
  int *penalty;
  float renapert,renideal;
  float maxinitburn;
```

```
{
 double xtfdot, ytfdot;
 double xcfdot, ycfdot;
 double xretro, yretro;
 double dv;   /* TOTAL DIFFERENCE IN VELOCITY BETWEEN
            TARGET AND CHASE CRAFTS      */

 double vburn;  /* VELOCITY OF THE CHASE CRAFT, NOT
             INCLUDING VELOCITY MATCHING DELTA V */
 double burnangle;  /* ANGLE OF BURN IN DEGREES */
 double renangle;   /* ANGLE OF RENDEZVOUS  */


 targpost = orbpnt(newset, targin->xpost,targin->ypost, temptt);

 *outinfo = rendezvous (chasposi->xpost, -chasposi->ypost,
          targpost->xpost, -targpost->ypost, temptt,
          newsetc->angrt);


 xtfdot = targpost->delvt * _lcos(targpost->alpt);
 ytfdot = -targpost->delvt * _lsin(targpost->alpt);

 xcfdot = (*outinfo)->sfdot;
 ycfdot = -(*outinfo)->rfdot;

 xretro = xtfdot-xcfdot;
 yretro = ytfdot-ycfdot;

 burnangle = _latan2((*outinfo)->rodot,(*outinfo)->sodot) *180/3.1416;
 renangle = _latan2(yretro,-xretro) *180/3.1416;

 dv = _lsqrt(xretro * xretro + yretro * yretro);

 vburn = _lsqrt((*outinfo)->rodot * (*outinfo)->rodot +
                  (*outinfo)->sodot * (*outinfo)->sodot);
 if ((burnangle > 0. && burnangle < maxinitburn/2)  ||
       (burnangle < 0. && burnangle > -maxinitburn/2))
          *penalty = 0;
 else
   *penalty =5;

 if ((*penalty) == 0)
  if ((renangle +180. < renideal - renapert/2) ||
       (renangle + 180. > renideal + renapert/2))
     *penalty = 5;
```

```
                /* these lines of code were
                   written so that they would
                   match navie. This is so the
                   burn is penalized if it does not
                   fall in the cone at rendez */

   return(dv + vburn + *penalty);
  }

main (argc, argv)
 int argc;
 char *argv[];

{
 static char chasestr[] = { "chase.dat" };

 int index;          /* COUNTER VARIABLE */
 int lfinal,lpresent;   /* NUMBER OF ACCEPTED TRANSITIONS BEFORE ITERATING
                  THE CONTROL PARAMETER, AND NUMBER OF ALREADY
                  ACCEPTED TRANSITIONS   */
 double czero,cfinal,cupdate; /* VALUES FOR THE CONTROL PARAMETER:
                     INITIAL,FINAL, AND CURRENT  */
 double decrement;       /* CONTROL PARAMETER AT STEP K IS
                  DECREMENTED BY THIS FRACTION  */
 int check;           /* BOOLEAN TO DETERMINE IF PRESENT ITERATION
                  IS THE FIRST OR NOT  */
 int numruns;         /*TOTAL NUMBER OF SAMPLES TO FIND MIN */

 double tt,temptt;     /* RELATIVE TIME TO COMPLETE RENDEZVOUS */
 double flat_time,temp;
 double new_v,temp_v1,temp_v2;
 double v;          /* TOTAL BURN REQUIRED */
 double delcost;       /* TOTAL CHANGE OF DELTA V FROM ONE
                  ITERATION TO THE NEXT       */
 double vprev;         /* DELTA V OF THE PREVIOUS ITERATION
                  COST (FUEL COST)          */

 float renideal,renapert; /* ANGLES FROM NAVIE INPUT
                  FILE DESCRIBING THE ARRIVAL ANGLE AND
                  THE APERTURE OF THE ACCEPTABLE CONE */

 float maxinitburn; /* APERATURE IN DEGREES OF ALLOWABLE BURN
                AT INITIATION       */

 double bestyet;       /* HOLDS MIN VALUE TO PRESENT */
 int penalty;      /* PENALTY FOR VIOLATING A CONSTRAINT */
```

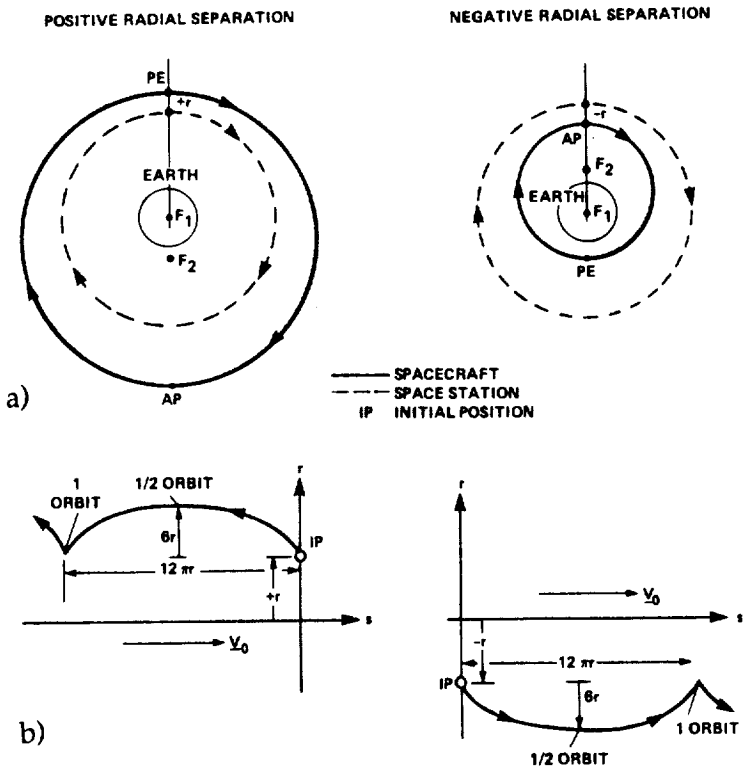
```
int spacepen;        /* PENALTY FOR VIOLATING SPATIAL CONSTRAINT */
double v_no_spen;  /* FUEL USED INCLUDING PENALTIES BUT THE
                SPATIAL PENALTY          */

double clockcheck,test;
struct position *targpost;    /* TARGET CRAFT POSITION AT TIME T */
struct position *chasposi;    /* CHASE CRAFT POSITION AT TIME 0 */

struct burninfo *outinfo;     /* BURN COMPONENTS AND DERIVATIVES */
struct position *targin;      /* HOLDS TARGET CRAFT INPUT DATA */
struct position *chasein;     /* HOLDS TARGET CRAFT INPUT DATA */

struct orbset *newset;        /* HOLDS ORBITAL PARAMETER SET
                                        FOR TARGET CRAFT       */
struct orbset *newsetc;       /* HOLDS ORBITAL PARAMETER SET
                                    FOR CHASE CRAFT          */


if (argc != 6)
 {
 printf("SPECIFY 5 ITEMS ON THE COMMAND LINE\n");
 printf("    targetfile czero cfinal chainaccepts decrement\n");
 exit(1);
 }

TIME0
targin = rdfile(argv[1]);
chasein = rdfile(chasestr);

sscanf(argv[2], "%lf",&czero);
sscanf(argv[3], "%lf",&cfinal);
sscanf(argv[4], "%d", &lfinal);
sscanf(argv[5], "%lf",&decrement);

if (decrement >= 1.0 || decrement <= 0.0)
  {
   printf("DECREMENT VALUE MUST BE BETWEEN 0 AND 1\n");
   exit(1);
  }
if ( czero < cfinal || cfinal < 0)
 {
  printf("CONTROL PARAMETER VIOLATION: BE SURE CZERO > CFINAL > ZERO\n");
  exit(1);
 }
vprev = 0.0;
numruns =0;
cupdate=czero;
```

```
check=1;
renideal = 60.;
renapert = 40.;
maxinitburn = 160.;

bestyet = 1000;     /* absurd high value */
srand48(time((long*) 0));

do
  {
    lpresent=0;  /* resets accepted transitions to zero */
    while( lpresent < lfinal)
    {
     do
      {
       times(&t_timbuf);
       clockcheck=(t_timbuf.tms_utime - t_ltime) / HZ ;
       if (clockcheck > 100.)
          { printf("\n clockcheck violation!!  %d\n", numruns);
            printf("last iteration: %4.2lf best %4.2lf\n", v, bestyet);
            TIME1(time use)
            exit(6);
          }
       if (check)
         temptt = 90.0 * drand48();

       else
          temptt = tt + 10. * (drand48() - 0.5); /* transition values */

      }
      while( temptt > 90. || temptt < 0 );

      newset = orbpar(targin, 0);
      newsetc = orbpar(chasein, 0);

      chasposi = orbpnt(newsetc, chasein->xpost,chasein->ypost, 0);
      v_no_spen = space_calc(newset,newsetc,targin,chasein,
              temptt,chasposi, targpost,&outinfo,&penalty,
              renapert, renideal, maxinitburn);
      if (penalty == 0)
        spacepen = spatial(penalty,chasein,outinfo,tt);
      v = v_no_spen + spacepen;


      if (v < bestyet)
         bestyet =v;
```

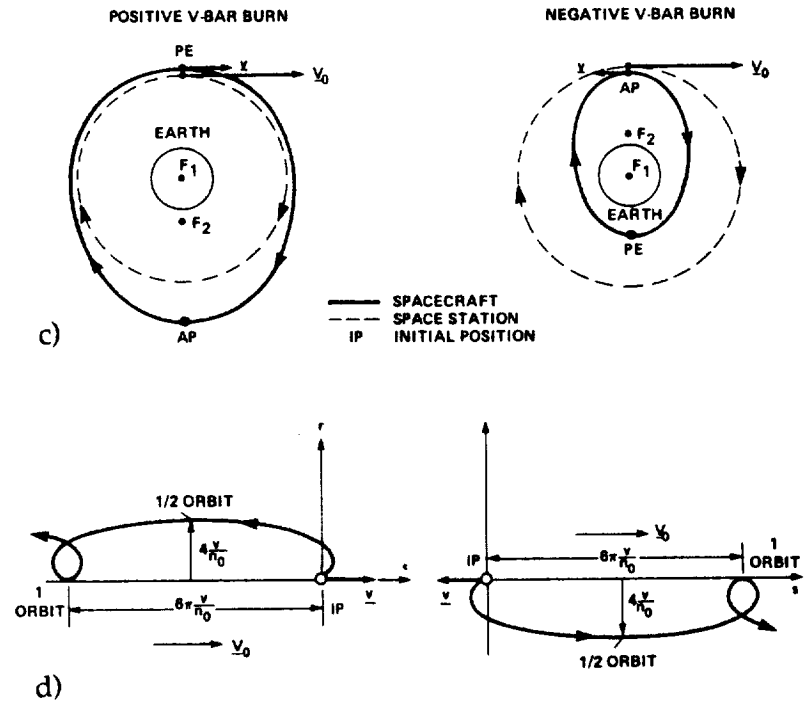
```
        delcost= v-vprev;


        if (delcost <= 0.)
         {
          lpresent +=1;
          tt=temptt;
          vprev= v;
         }
        else if ( _lexp(-delcost/cupdate) > (test =drand48()) )
         {
          lpresent +=1;
          tt= temptt;
          vprev= v;
         }
        else if (check==1)
         {
          tt=temptt;
          vprev= v;
         }

          check=0.;
          numruns +=1;
        /*   if (numruns > 4000)
              {
  printf("ck %.5lf l %d tt %4.2lf v %4.2lf delcost %4.2lf best %4.2lf run %d\n",
cupdate,lpresent,tt,v,delcost,bestyet,numruns);

              exit(1);
             } */
        }
       cupdate *= decrement;
      }
    while(cupdate >= cfinal);
    if ( spacepen == 5 || penalty == 5)
       printf("\nNO UNPENALIZED TRAJECTORY FOUND!\n");
    if ( spacepen == 5)
      printf("Spatial constraint violated \n");

    flat_time = tt -0.5;
    new_v = v;

     while ( new_v - v <= .03  || new_v < v )
      {
       temp_v1 = new_v;
       new_v = space_calc(newset,newsetc,targin,chasein,
```

```
                flat_time,chasposi, targpost,&outinfo,&penalty,
                renapert, renideal, maxinitburn) +
            spatial(penalty,chasein,outinfo,flat_time);
        flat_time += -.5;
        }

    temp = flat_time + 0.5;

    flat_time = tt + 0.5;
    new_v = v;

    while( new_v - v <= 0.03 || new_v < v)
      {
       temp_v2 = new_v;
       new_v = space_calc(newset,newsetc,targin,chasein,
                flat_time,chasposi, targpost,&outinfo,&penalty,
                renapert, renideal, maxinitburn) +
            spatial(penalty,chasein,outinfo,flat_time);
       flat_time += 0.5;
      }


    flat_time -= 0.5;
    if ( tt  > temp || flat_time > tt  )
       {
        printf("\n    FLAT OPTIMAL SEGMENT! \n");
        printf("     beginning of segment:  fuel %5.2lf  time %5.2lf\n",
              temp_v1, temp);
        printf("     end of segment:       fuel %5.2lf  time %5.2lf\n\n",
              temp_v2, flat_time);
       }


    printf("    %4.2lf %4.2lf %d %4.2lf %5.2lf %d %5.2lf %5.2lf\n",
              czero,cfinal,lfinal,decrement,v,numruns,tt,bestyet);
    printf("\n\n");
    TIME1(time to run optimization)
    printf("\n");
}
```

Figure 1



Orbital motion for an initially stationary spacecraft with radial separation r a) Shape of orbit b) Trajectory relative to the space station.



Orbital motion after a V-bar burn V c) Shape of orbit d) Trajectory relative to the space station.

FIGURE 2

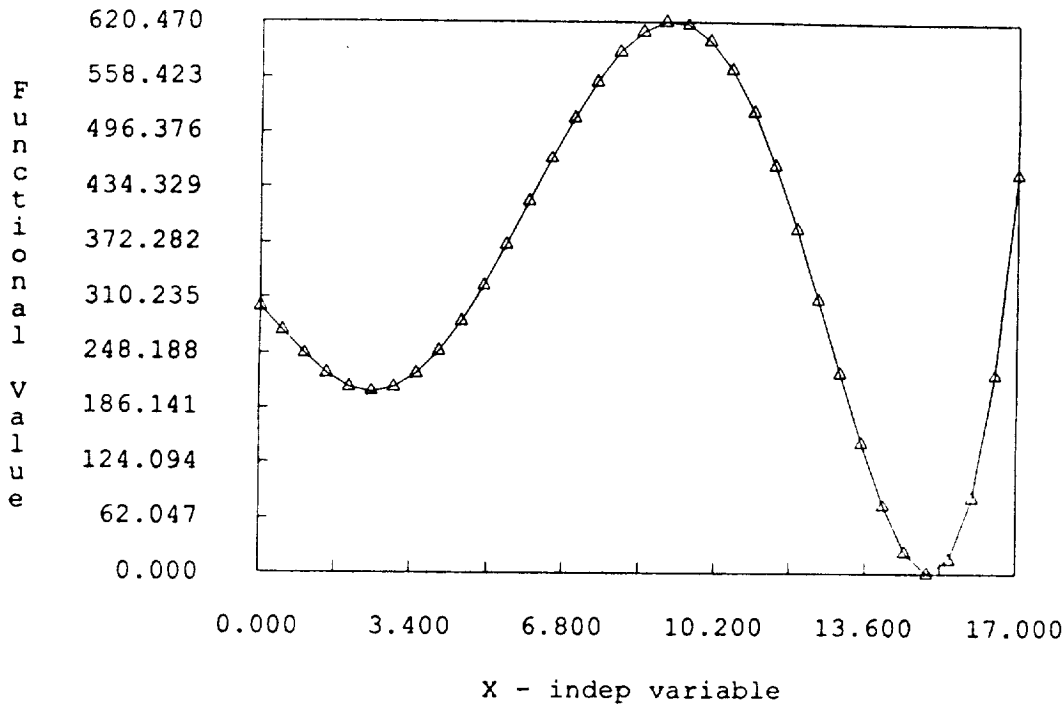
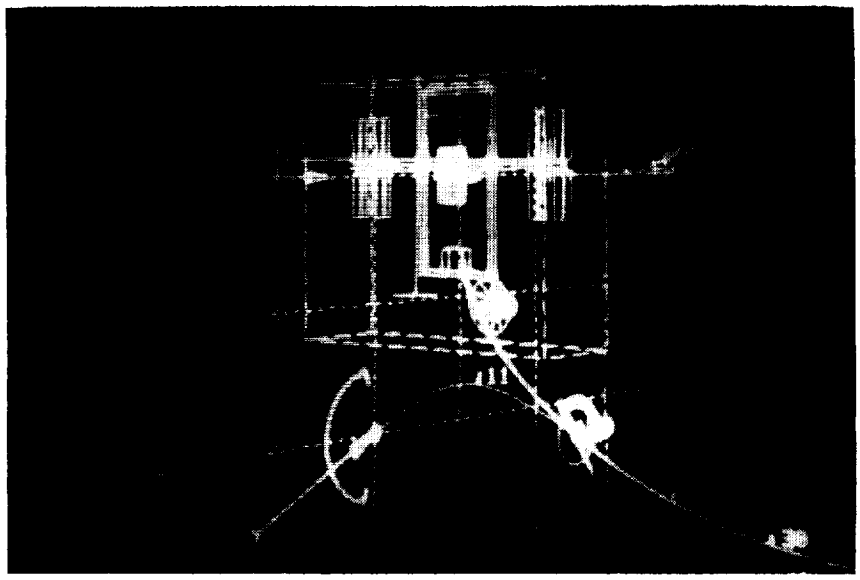

620.470
558.423
496.376
434.329
372.282
310.235
248.188
186.141
124.094
62.047
0.000
Functional Value
0.000
3.400
6.800
10.200
13.600
17.000
X - indep variable

Figure 3



Explanation:

1. The dotted grid represents the orbital plane.
2. The dashed region surrounding the space station represents the spatial contraint.
3. The target craft starts on the bottom section of the space station keel.
4. The chase craft starts below spatial constraint.
5. The arc in the area of the chase craft represents an angular departure constraint.
6. The lines extending from the chase and target crafts represent their trajectories.
7. The numbers on the trajectory lines describe the position of the craft at a specified number of minutes into the mission.
8. Rendezvous, denoted by a circle occurs at 20 minutes.

FIGURE 4

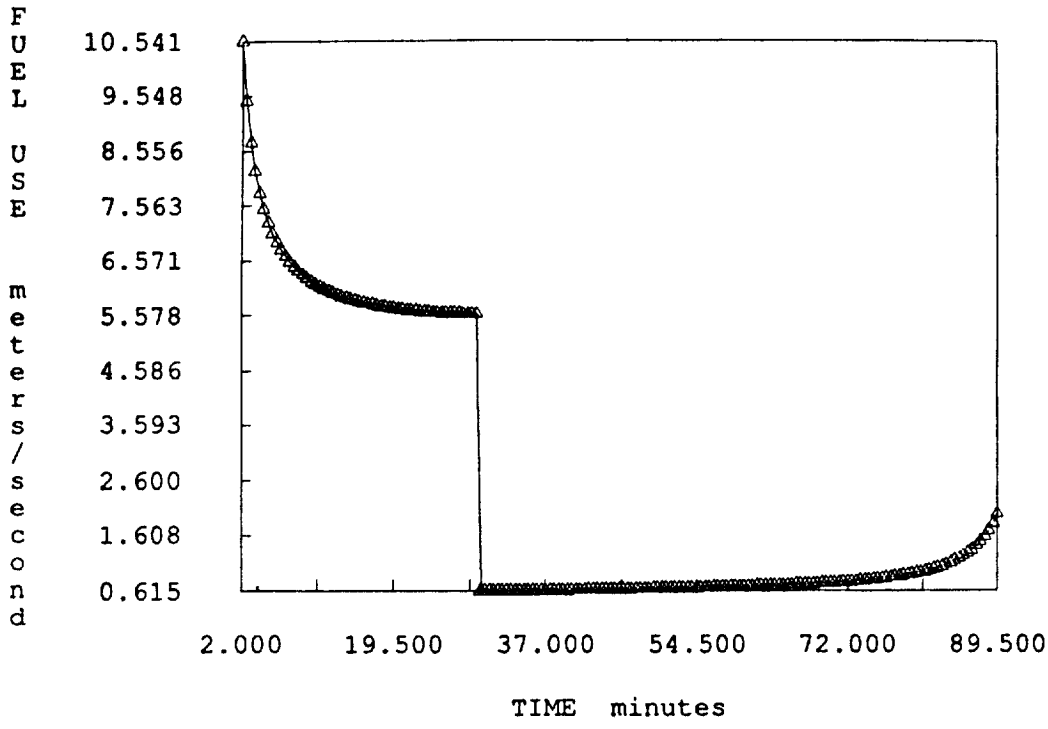


FIGURE 5

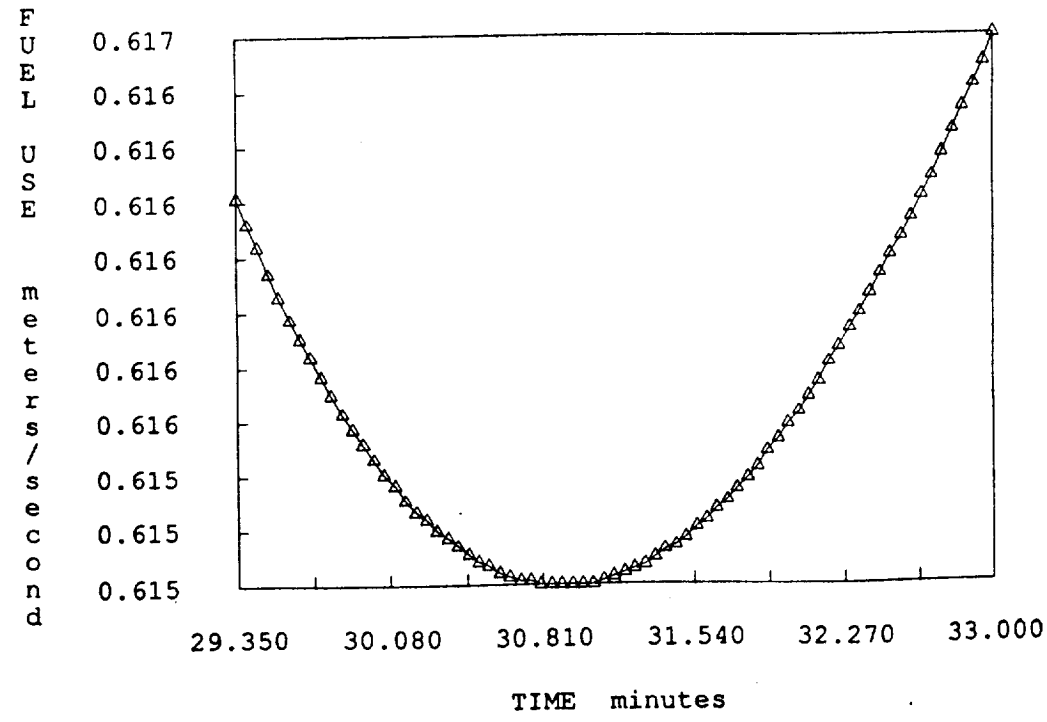
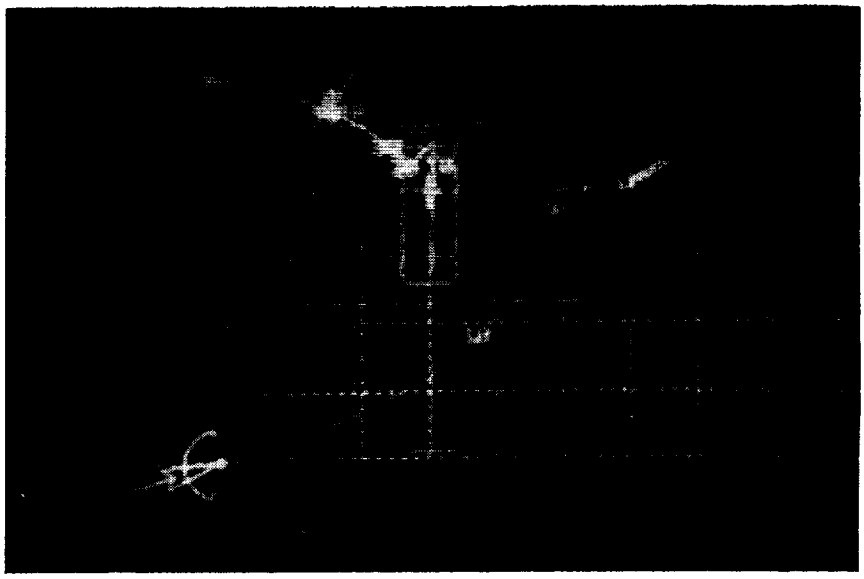


Figure 6



Explanation:

1. The dotted grid represents the orbital plane.
2. The dashed region surrounding the space station represents the spatial contraint.
3. The target craft starts on the top section of the space station keel.
4. The chase craft starts below spatial constraint.
5. The arc in the area of the chase craft represents an angular departure constraint.
6. The lines extending from the chase and target crafts represent their trajectories.
7. The numbers on the trajectory lines describe the position of the craft at a specified number of minutes into the mission.
8. Rendezvous, denoted by a circle occurs at 40 minutes.

FIGURE 7

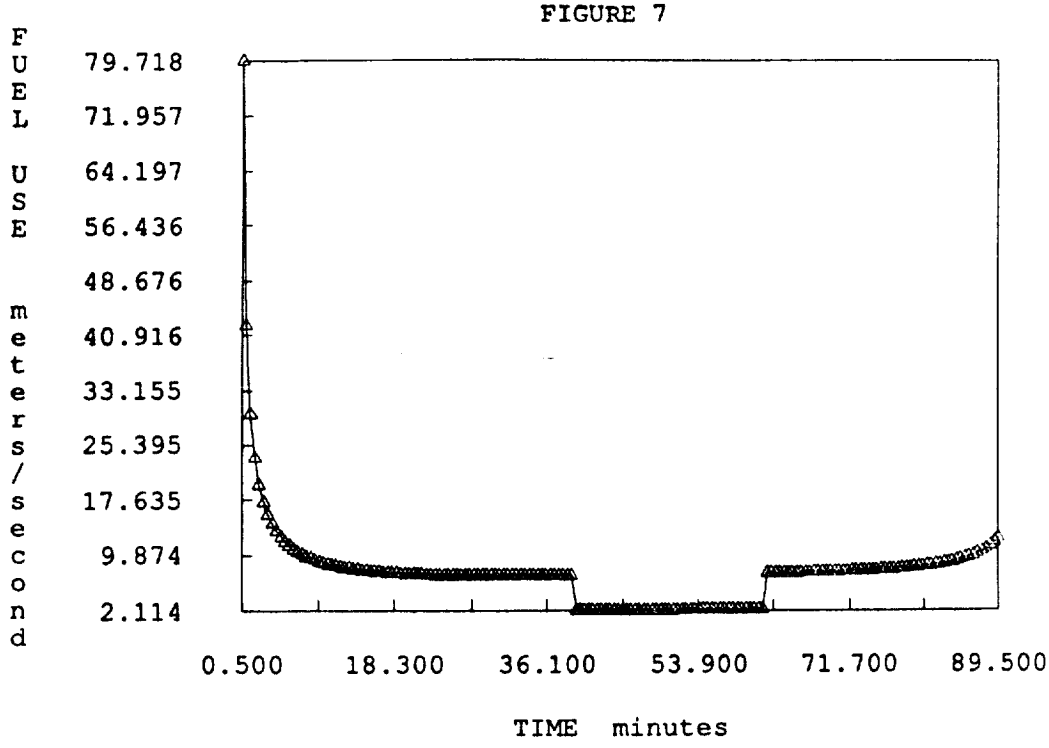


FIGURE 8

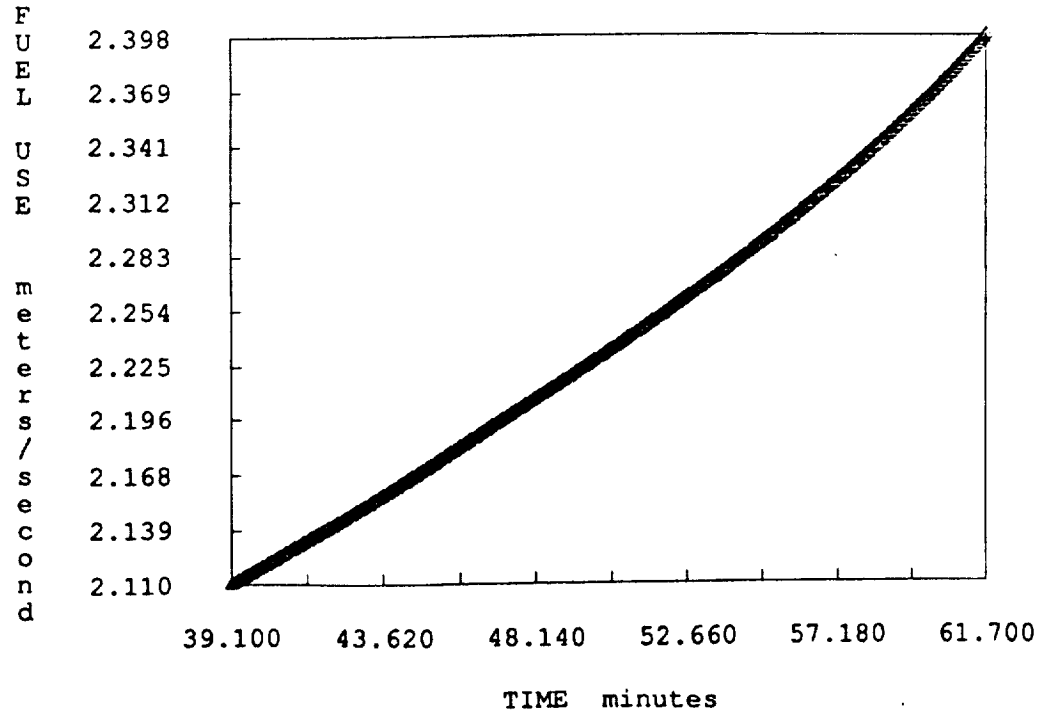
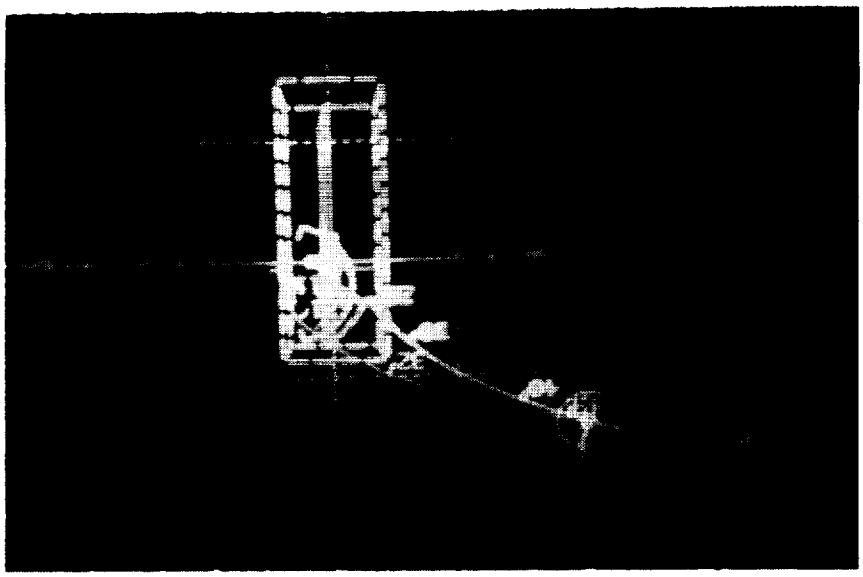


Figure 9



Explanation:

1. The dotted grid represents the orbital plane.
2. The dashed region surrounding the space station represents the spatial contraint.
3. The target craft starts on the top section of the space station keel.
4. The chase craft starts ahead of the station in the spatial constraint.
5. The arc in the area of the chase craft represents an angular departure constraint.
6. The numbers on the trajectory lines describe the position of the craft at a specified number of minutes into the mission.
7. Rendezvous, denoted by a circle occurs at 49 minutes.

FIGURE 10

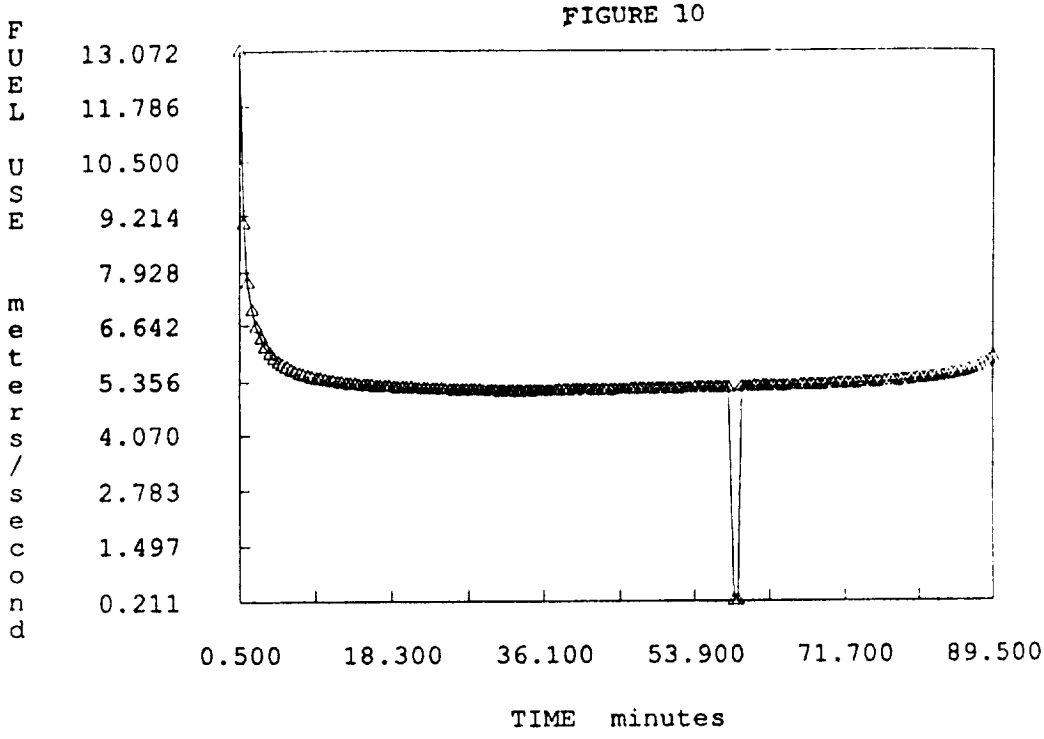


FIGURE 11

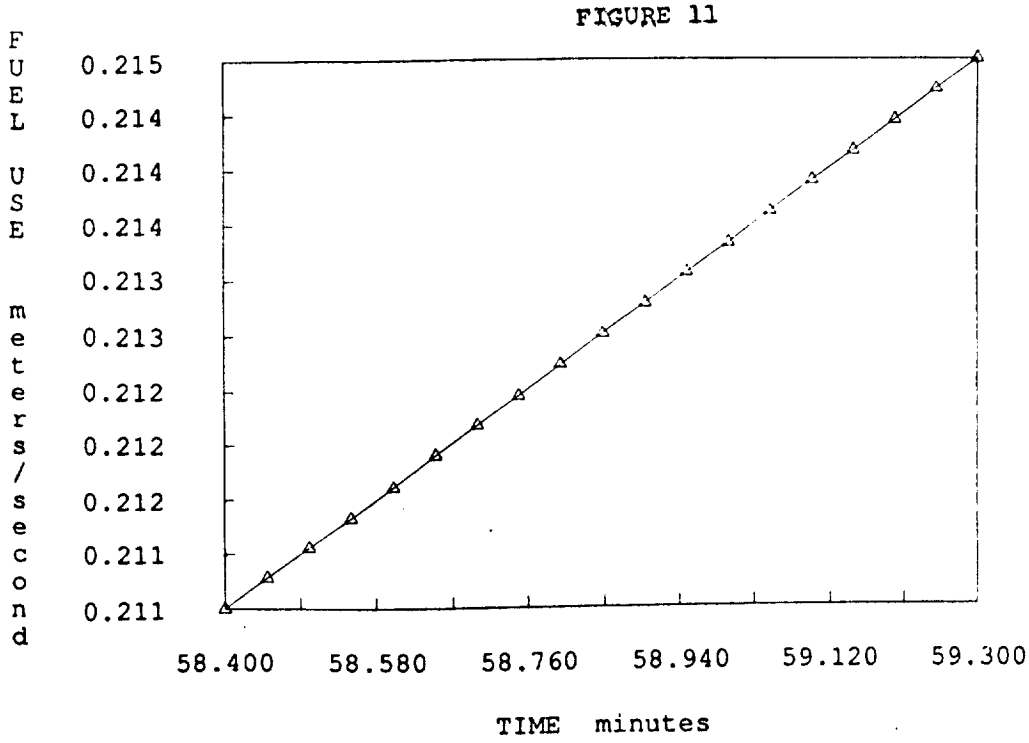
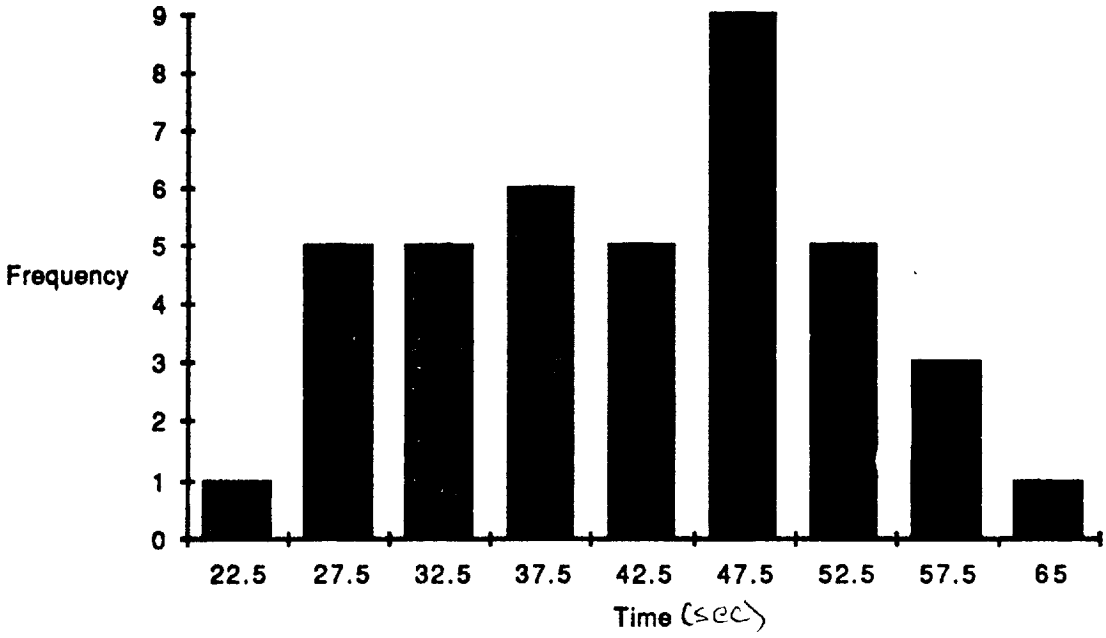


Figure 12


Frequency
9
8
7
6
5
4
3
2
1
0
22.5
27.5
32.5
37.5
42.5
47.5
52.5
57.5
65
Time (sec)

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-102866		

4. Title and Subtitle	5. Report Date
A Trajectory Planning Scheme for Spacecraft in the Space Station Environment	January 1991
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Jeffrey Alan Soller, Arthur J. Grunwald, and Stephen R. Ellis	A-90287
	10. Work Unit No. 506-47-31
9. Performing Organization Name and Address	11. Contract or Grant No.
Ames Research Center Moffett Field, CA 94035-1000	
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Memorandum
National Aeronautics and Space Administration Washington, DC 20546-0001	14. Sponsoring Agency Code

15. Supplementary Notes

Point of Contact: Jeffrey A. Soller, Ames Research Center, MS 239-3, Moffett Field, CA 94035-1000
(415) 604-6147 or FTS 464-6147

Originally presented as a thesis for an MS degree in the Department of Industrial Engineering Operations Research at the University of California at Berkeley.

16. Abstract

Simulated annealing is used to solve a minimum fuel trajectory problem in the space station environment. The environment is special because the space station will define a multivehicle environment in space. The optimization surface is a complex nonlinear function of the initial conditions of the chase and target crafts. Small permutations in the input conditions can result in abrupt changes to the optimization surface. Since no prior knowledge about the number or location of local minima on the surface is available, the optimization must be capable of functioning on a multimodal surface. It has been reported in the literature that the simulated annealing algorithm is more effective on such surfaces than descent techniques using random starting points.

The simulated annealing optimization was found to be capable of identifying a minimum fuel, two-burn trajectory subject to four constraints which are integrated into the optimization using a barrier method. The computations required to solve the optimization are fast enough that missions could be planned on board of the space station. Potential applications for on board planning of missions are numerous. Future research topics may include optimal planning of multi-waypoint maneuvers using a knowledge base to guide the optimization, and a study aimed at developing robust annealing schedules for potential on board missions.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Proximity operations Trajectory optimization Simulated annealing	Unclassified-Unlimited Subject Category - 12

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	58	A04

NASA FORM 1626 OCT 86

For sale by the National Technical Information Service, Springfield, Virginia 22161